モデルコンパイラのアーキテクチャの自己反映適用

2014SE004 青木公宏　2014SE056 倉地亮介
指導教員：野呂昌満

1　はじめに

モデル駆動工学 (Model-Driven Engineering: MDE) において，モデルコンパイラの作成の省力化は重要な課題である [1]．モデルコンパイラは，入出力であるテキスト・グラフによって 4 つに分類できる．グラフを表現したテキストを分析し構文木を生成するテキスト–グラフ変換，入力モデルを目的モデルに変換するグラフ–グラフ変換，目的モデルをテキストに変換するグラフ–テキスト変換，及びテキスト–テキスト変換の 4 つである．テキスト–テキスト変換は，テキスト–グラフ変換とグラフ–テキスト変換を組み合わせることで表すことができるので，実質 3 種類のモデルコンパイラを作成する必要がある．江坂 [1] らは，これらのモデルコンパイラすべてに統一的なモデルコンパイラのための統一アーキテクチャを提案している．多岐にわたるプラットフォームごとのモデルコンパイラを各個に開発するとその労力は膨大になる [4]．我々は，解決方法としてモデルコンパイラの自動生成を考える．言語プロセッサの自動生成については，既に実用化された Yacc，JavaCC などが存在する．これらは，主にソーステキストを入力とし，オブジェクトコードを出力とすることを念頭に置いている．この自動生成技術を基礎にモデルコンパイラを自動生成するメタモデルコンパイラの実現を考える．

本研究の目的は，メタモデルコンパイラを実現することである．メタレベルとベースレベルでの共通なアーキテクチャを設計する．すなわち，ベースレベルのモデルコンパイラの統一アーキテクチャを自己反映的にメタモデルコンパイラに適用し，統一的なアーキテクチャでメタモデルコンパイラを実現する．

本研究では，メタレベルとベースレベルで共通なアーキテクチャを設計し，アーキテクチャの実現に向けてグラフ変換の新たな文法を提案した．メタモデルコンパイラとベースレベルの 3 種類のモデルコンパイラのアーキテクチャを整理する．モデルコンパイラのための統一アーキテクチャを改版し，アスペクト指向技術を用いてメタレベルとベースレベルに共通なモデルコンパイラのアーキテクチャを設計する．グラフ変換の新たな文法を提案し，生成規則に優先順位を与える処理アルゴリズムを考え処理系を試作した．

2　背景技術

モデルコンパイラのアーキテクチャ及びグラフ変換を記述するグラフ文法について述べる．

2.1　モデルコンパイラのための統一アーキテクチャ

江坂らは，モデルコンパイラのための統一アーキテクチャ (図 1) を設計した．上記 3 種類のモデルコンパイラの複数の関心事をアスペクト指向技術により，1 つのアーキテクチャ上で表現した．すなわち，入力や出力を指定することで特定のアーキテクチャを導出できるメタアスペクト指向アーキテクチャとして定義した．我々はこれを統一的なアーキテクチャと呼ぶ．このアーキテクチャは，グラフ–グラフ変換のために一般化されたパターンを採用 [2] し，Parsing Aspect，Graph Analysis Aspect，De-parsing Aspect，Graph Construction Aspect の 4 つのアスペクトモジュールを織り込むことによって，テキスト処理とグラフ処理の 2 つの処理を 1 つのアーキテクチャに統合し，テキスト–グラフ変換及びグラフ–テキスト変換を考慮できるアーキテクチャである．

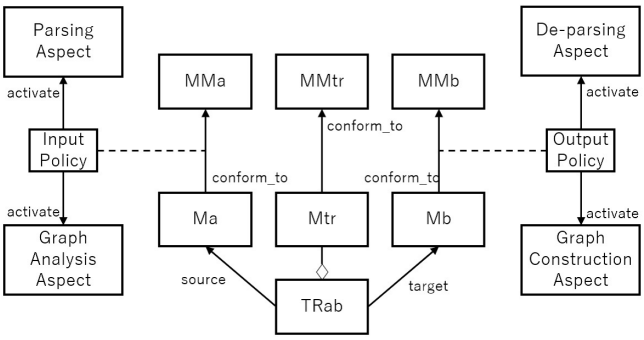

図 1　モデルコンパイラのための統一アーキテクチャ [1]

- Ma: ソースモデル
- Mb: ターゲットモデル
- MMa: ソースモデルの概要シンタックス
- MMb: ターゲットモデルの概要シンタックス
- Mtr: 生成規則
- MMtr: 生成規則の概要シンタックス
- TRab: 変換ツール

1. テキスト–グラフ変換
   ソースモデル Ma はテキスト表記で記述する．Input Policy によって Parsing Aspect が活性化される，Parser が入力テキストを処理する．その後，Mtr に記述された生成規則に従ってモデル変換を行う．変換後のモデルは抽象構文木で表されたグラフであるので，Output Policy によって Graph Construction Aspect が活性化し，メタモデル MMb に従ってターゲットモデル Mb を生成する．

2. グラフ–グラフ変換
   ソースモデル Ma はグラフ表記で記述する．Input Policy によって Graph Analysis Aspect が活性化される，Parser が入力グラフの要素をチェックする．その後，Mtr に記述された変換ルールに従ってモデル変換を行う．変換後のモデルは抽象構文木で表されたグラフであるので，Output Policy によって Graph Construction Aspect が活性化し，メタモデル MMb に従ってターゲットモデル Mb を生成する．
3. グラフ–テキスト変換
   ソースモデル Ma はグラフ表記で記述するので，グラフ–グラフ変換と同様に入力グラフの要素をチェックする．その後，Mtr に記述された変換ルールに従ってモデル変換を行う．すなわち，Mtr に記述される変換ルールの Action による出力がテキストに変更されるだけであり，アーキテクチャとしては，グラフ–グラフ変換とグラフ–テキスト変換は同一なものとして扱える．本稿ではグラフ–グラフ変換のアーキテクチャと同一のものとして扱う．

2.2 グラフ文法 (Graph Grammar)

グラフ文法とは，グラフやグラフ状の構造を生成・解析・変換するための操作のメカニズムを提供するものである．グラフ文法 GG は四つ組で定義される [3].

$$GG = \{T, S, P, A\}$$

T はグラフのタイプまたはクラスである．S は開始グラフ (start graph) と呼ばれ，最初に生成規則が適用されるグラフである．P は生成規則の集合である．A は生成規則の適用条件を記述する．

グラフ文法は G_0 を初期グラフとし，生成規則の有限集合 $P = \{p_1, p_2, ..., p_n\}$ の各生成規則 p_i(i=1,2,...) は Left Side Graph(LSG) と Right Side Graph(RSG) を有する．LSG ⇒ RSG と表現された生成規則が開始グラフまたは生成規則により生成されたグラフ G_i(i=0,1,2,...) に適用されると，グラフ G_i 内の LSG の同型サブグラフを探索する．LSG の全ては，接続または結合操作を使用して RSG に置き換えられ，$G_{(n+1)}$ が生成される．

3 拡張統一アーキテクチャの設計

モデルコンパイラとメタモデルコンパイラの概略を示し，これらを統一に扱うアーキテクチャを設計する．

3.1 メタモデルコンパイラ

本研究では，Yacc など実用的な言語生成系に倣い，処理対象の処理プロセスを構造化データとして表現し，それを解釈・実行する処理系と一対にすることで生成系とする．生成系の生成では処理対象の構造化データを生成する．

メタモデルコンパイラは，テキスト–グラフ変換のモデルコンパイラである．メタモデルコンパイラは，目的の生成系に必要とされる解釈・実行を行うインタプリタを内包している．メタモデルコンパイラの入力は，構文規則や生成規則などの生成系の仕様記述であり，出力は処理対象の構造化データであるプッシュダウン・オートマトンまたは有限オートマトンである．すなわち，メタモデルコンパイラはテキストを入力として，グラフを出力する生成系である．メタモデルコンパイラのアーキテクチャを図 2 に示す．

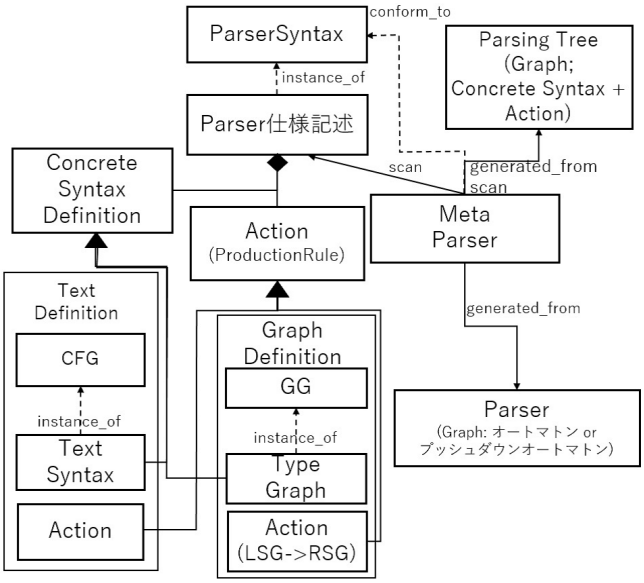

図 2 メタモデルコンパイラのアーキテクチャ

- Text Definition : テキスト–グラフ生成系の仕様記述
- Graph Definition : グラフ–グラフ生成系の仕様記述
- Meta Parser :文形式を処理，解析木を作成する．
- Action :生成規則の定義

メタモデルコンパイラのアーキテクチャは，テキスト–グラフ変換のアーキテクチャである．メタモデルコンパイラには，ベースレベルのテキスト–グラフ変換のアーキテクチャを自己反映的に適用することができる．ベースレベルの生成系のアーキテクチャを統一化することで，メタレベルとベースレベルで共通なモデルコンパイラのアーキテクチャを実現する．

3.2 生成系のアーキテクチャ

ベースレベルのモデルコンパイラの統一アーキテクチャを考える上で，テキスト–グラフ生成系とグラフ–グラフ生成系のアーキテクチャの概略を示し，2 つのアーキテクチャの相違点を整理する．図 3 に 2 種類の生成系のアーキテクチャの比較を示す．

我々は，2 種類の生成系のアーキテクチャを比較し，構造上の違いと処理を行う Parser の違いがあると考えた．構造上の違いについては，テキスト–グラフ生成系では，入力がテキストであるので，テキストからトークンリストを作成するコンポーネントが存在するが，グラフ–グラフ生成系では存在しない．処理を行う Parser の違いについて

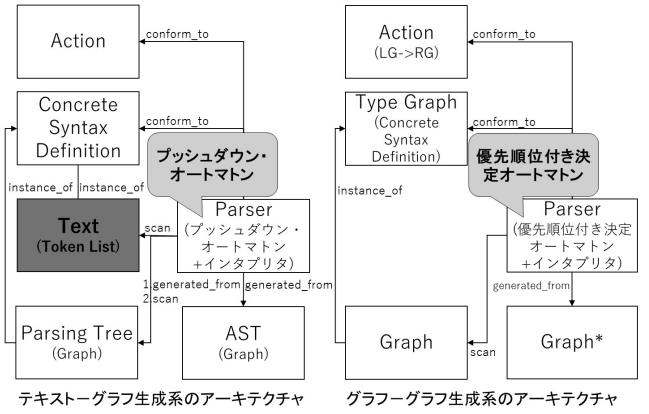

図 3　2 種類の生成系のアーキテクチャの比較

は，テキスト–グラフ生成系が扱う Parser のインタプリタはプッシュダウン・オートマトンを解釈対象とし，グラフ–グラフ生成系では，優先順位付き決定オートマトンを解釈対象とする．グラフ–グラフ変換の Parser は，複数の生成規則を持っており優先度を決め，優先順位通りに生成規則を適用してグラフ変換を行う必要がある．

3.3　アーキテクチャ設計指針

拡張統一アーキテクチャの設計指針を次のように定義する．

- モデルコンパイラのための統一アーキテクチャ [1] の改版としてアーキテクチャを設計
- アスペクト指向技術を用いてアーキテクチャを設計

本研究では，モデルコンパイラのための統一アーキテクチャを基にアーキテクチャを設計する．設計するアーキテクチャは，入出力の部分をテキスト・グラフの両方に対応可能な構造にし，入力に応じたアスペクトを取捨選択することによって，特定のアーキテクチャを導出することができるメタアーキテクチャとして定義する．

構造上の違いをアスペクト指向技術を用いて生成系のアーキテクチャの同一インタフェースで入れ替え可能にすることを目指す．入力に応じてアスペクトであるトークンリストを作成するコンポーネントを付加するかしないかを表現することで，テキスト–グラフ生成系とグラフ–グラフ生成系の構造上の違いを柔軟に対応できるアーキテクチャを実現できると考える．

処理を行う Parser の違いについては，それぞれの Parser のインタプリタが解釈する対象に着目し，処理方法の統一化を目指す．グラフ–グラフ変換のインタプリタの解釈対象である優先順位付き決定オートマトンで扱う正規文法 (Regular grammar) は，プッシュダウン・オートマトンで扱う文脈自由文法 (Context-free grammar) の特殊形であり，両方とも Context-free grammar で表現するという枠組みで，Parser の処理方法を同一インタフェースで入れ替え可能とする．

3.4　提案するアーキテクチャの概略

江坂 [1] らが提案したモデルコンパイラのための統一アーキテクチャを用いて，以下の 3 つのアスペクトモジュールを追加した．図 4 は，本研究で提案するアーキテクチャの概略である．

- Scanner: トークンリストを作成し，走査する
- Precedence: Parser に生成規則の優先順位を与える
- Reduction: 文形式を処理する

2 つの生成系のアーキテクチャの構造上の違いは，トークンリストの有無である．提案するアーキテクチャでは，トークンリストを作成するコンポーネントを Scanner アスペクトとする．入力がテキストである場合は，Scanner アスペクトが付加され，トークンリストを作成する．入力がグラフである場合は，Scanner アスペクトは付加されない．提案するアーキテクチャは，入力に応じたアスペクトの付加により柔軟にテキスト–グラフ生成系またはグラフ–グラフ生成系のアーキテクチャの構造に変身できるという点で統一的なアーキテクチャである．

Parser の処理方法では，Parser の構造内でアスペクト指向技術を用いることで処理方法の違いを統一化した．提案するアーキテクチャでは，Parser の構造内で Precedence アスペクトと Reduction アスペクトを用意する．入力がテキストである場合は，Reduction アスペクトが付加され，インタプリタの解釈対象がプッシュダウン・オートマトンとなる．入力がグラフである場合は，Precedence アスペクトが付加され，インタプリタの解釈対象が優先順位付き決定オートマトンとなる．提案するアーキテクチャは，入力に応じてインタプリタの解釈対象を柔軟に変更できるという点で統一的なアーキテクチャである．

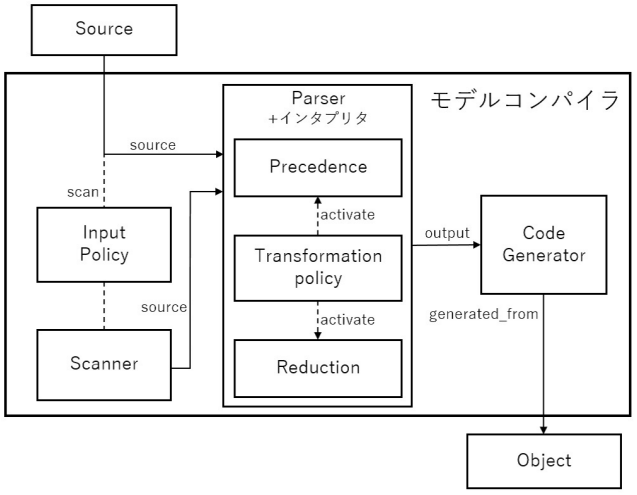

図 4　本研究で提案するアーキテクチャの概略

4　アーキテクチャに基づくグラフ文法の処理系

アーキテクチャに基づくグラフグラフ変換のメタグラフ文法を提案し，その処理系を試作する．

4.1 グラフ文法の拡張

優先順位付き決定オートマトンのクラスの文法であるグラフ文法の軌道を定義するメタグラフ文法を Precedence Graph Grammar として提案する．本研究では，Parra[3] によって定義されているグラフ文法のメタレベルに演算子順位文法 (Operator Precedence Grammar)[5] を適用することで優先順位を表現するメタグラフ文法を定義する．
演算子順位文法の記法に従って Precedence Graph Grammar は

$$PGG = \{SG, Vt, mP\}$$

で定義する．SG はサブグラフの集合，Vt は終端記号の集合，mP は生成規則の集合，⇒は変換処理を表すオペレータである．以下に記述例を示す．

$$PGG1 = \{\{SG_1, SG_2, SG_3, SG_4\}, \{\Rightarrow^1, \Rightarrow^2, \Rightarrow^3\}, mP\}$$
$$mP = \{P1 : SG_1 \Rightarrow^1 SG_2$$
$$P2 : SG_2 \Rightarrow^2 SG_3$$
$$P3 : SG_3 \Rightarrow^3 SG_4$$
$$\}$$

変換処理を表すオペレータの優先度の関係を表 1 のように示す．

表 1　生成規則の優先順位表

	$\Rightarrow^1$	$\Rightarrow^2$	$\Rightarrow^3$
$\Rightarrow^1$	$\doteq$	$\gtrdot$	$\gtrdot$
$\Rightarrow^2$	$\lessdot$	$\doteq$	$\gtrdot$
$\Rightarrow^3$	$\lessdot$	$\lessdot$	$\doteq$

4.2 Precedence Graph Grammar の処理アルゴリズム

メタグラフ文法の処理アルゴリズムを考察し，処理系を試作した．Parra によって定義されているグラフ文法では，生成規則の優先順位を処理するアルゴリズムは提案されていない．Precedence Graph Grammar で定義した生成規則の優先順位表を基に生成規則を優先順位通りにソーティングを行い，優先度が高いオペレータから順に適用する．

5　考察

5.1　アーキテクチャの自己反映適用に関する考察

本研究では，メタモデルコンパイラはテキスト–グラフ変換のアーキテクチャであることを示し，ベースレベルのモデルコンパイラのアーキテクチャをメタレベルでも自己判定的に適用できると考えた．ベースレベルの 3 種類のモデルコンパイラすべてに統一的なアーキテクチャを設計し，メタレベルでの適用も可能にすることでメタモデルコンパイラのアーキテクチャを実現する．モデルコンパイラを安直に開発した場合，モデルコンパイラの種類の数だけ作成方法が存在し，開発は困難になると考える．本研究で提案したアーキテクチャを用いることで，メタレベルとベースレベルでのモデルコンパイラ開発の仕組みを 1 つにすることができ，モデルコンパイラ開発を簡素化できる．

5.2　提案したアーキテクチャの実現に向けての考察

本研究では，優先順位付き決定オートマトンのクラスの文法であるグラフ文法の軌道を定義するメタグラフ文法を Precedence Graph Grammar として提案した．メタグラフ文法の処理アルゴリズムを考察し，処理系を試作した．処理系の試作により，メタグラフ文法の処理アルゴリズムでソーティングを行い優先順位通りに生成規則を適用できることを確認した．Precedence Graph Grammar の処理アルゴリズムを用いることで，グラフ文法は優先順位を考慮した生成規則の適用を行うグラフ–グラフ生成系の Parser を実現できると考える．

6　おわりに

本研究では，モデルコンパイラの開発支援を目的とし，メタモデルコンパイラを実現するためにメタレベルとベースレベルでのモデルコンパイラに共通なアーキテクチャを設計した．テキスト–グラフ変換とグラフ–グラフ変換での構造上の違いと Parser の処理方法の違いを整理し，アスペクト指向技術を用いることでテキスト–グラフ変換とグラフ–グラフ変換を 1 つのアーキテクチャで統一的に扱えることを示した．グラフ変換の新たな文法を提案し，アーキテクチャの実現に向けての考察を行った．設計したアーキテクチャを用いることで，メタモデルコンパイラが実現でき，モデルコンパイラ作成の省力化に貢献できると考える．

参考文献

[1] A. Esaka, M. Noro, and A. Sawada, “Design of Common Software Architecture as Base for Application Generator and Meta-Generator for Interactive Systems,” *IEEE*, vol. 2, pp. 323-328, 2017.

[2] F. Jouault, F. Allilaire, J. Bezivin, and I. Kurtev, “ATL: A model transformation tool,” *Science of Computer Programming*, vol. 72, no. 1-2, pp. 31-39, 2008.

[3] F. Parra, “Application of Graph Grammars to Model Transformations,” *Technical Report*, 2013.

[4] 鵜林尚靖，佐野慎治，前野雄作，村上聡，片峯恵一，橋本正明，玉井哲夫，“アスペクト指向に基づく拡張可能な MDA モデルコンパイラ，” 組込みソフトウェアシンポジウム ESS2004，pp.104-107，2004.

[5] 中田育男，コンパイラの構成と最適化，朝倉書店，1999.